12

NUWC-NPT Reprint Report 10,591
14 February 1994

Massively Parallel Systems Design for Real-Time Embedded Applications

Thomas C. Cholnski
Submarine Sonar Department

Chin-Hwa Lee
Naval Postgraduate School

Naval Undersea Warfare Center Division
Newport, Rhode Island

Approved for public release; distribution is unlimited.

Reprint of a paper presented at the *Systems Reengineering Technology Workshop*, Monterey, California, 8-10 February 1994.

Systems Reengineering Technology Workshop, February 8-10, 1994

AN EFFICIENT APPROACH TO SYSTEMS EVOLUTION

Massively Parallel Systems Design for Real-Time Embedded Applications

Thomas C. Choinski and Chin-Hwa Lee

Naval Undersea Warfare Center
Newport Division
New London, CT 06320

Naval Postgraduate School
Monterey, CA 93943

Abstract

This paper describes a generic approach to mitigate risk when reengineering for high throughput massively parallel systems. The approach entails baselining the existing system, capturing the functional requirements, estimating initial processing requirements through a high level analysis, benchmarking a subset of the functionality on a low throughput computer, and modeling the high throughput application to determine the detailed processing requirements for scaling.

1: Introduction

"Once the architecture begins to take shape, the sooner contextual constraints and sanity checks are made on assumptions and requirements, the better."

Eberhardt Rechtin, Systems Architecting: Creating & Building Complex Systems [1]

Commercial massively parallel processing (MPP) architectures offer a solution to TERAFLOP (one trillion operations per second) computing applications in the Navy. Computing density (TERAFLOP/cubic foot) and cost (dollars/TERAFLOP) have decreased in recent years; however, the challenge of real-time embedded processing requirements poses a high risk for complex systems. The high risk relates to: inefficient match of applications to architectures, low availability of high throughput architectures, the accuracy of forecasted downward spiraling price projections, and immature software development tools (e.g., parallelizing compilers).

This paper proposes a generic approach to mitigate the risk when investing in a specific MPP architecture. The approach proposes a series of intermediate steps to assess the compatibility of the architecture and the requirements. Each step refines the assessment and leads to the final tradeoff study. The approach embodies reengineering considerations when pursuing new implementations for cost or technology upgrades. Specifically, the approach entails:

1. defining the system requirements,
2. sizing an architecture using static benchmarks,
3. allocating resources using systems engineering tools,
4. developing a full scale model,
5. validating the full scale model with dynamic benchmarks,
6. assessing the compatibility of the architecture with the real-time embedded applications, and
7. selecting the appropriate design approach based on a trade-off analysis.

The tradeoff analysis incorporates an assortment of design tools to expedite and facilitate the decision making process. Examples of tools used to date include: VHSIC Hardware Description Language (VHDL), RDD-100 and OMTool. The approach will integrate the systems engineering tools developed under the direction of the Naval Surface Warfare Center (NSWC) within the Office of Naval Research's (ONR) Engineering of Complex Systems (ECS) Block Program as they become available.

The generic approach suits a myriad of applications ranging from radar to sonar systems. Accordingly, air, surface and subsurface platforms can benefit from the approach outlined in this paper. This paper uses the case study method to showcase the approach.

The case study method applies the generic approach to a practical application. This paper discusses one case study to demonstrate the utility of the approach. The research will explore additional case studies as interest arises.

This paper describes the case study, outlines the generic systems engineering design approach, presents high level architectural sizing techniques, and discusses detailed modeling and requirements allocation issues. The

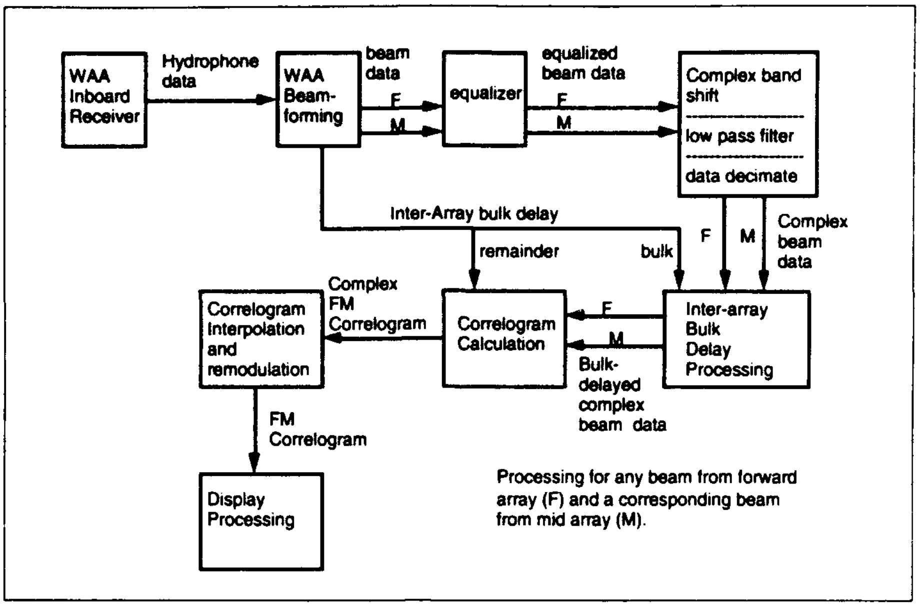

Figure 1. WAA Processing

Wide Aperture Array (WAA) beamforming problem serves as the initial case study for the application of the generic approach. The approach will evolve based on the comments, suggestions and progress of research performed in the ECS Program under the direction of the Naval Surface Warfare Center (NSWC).

2: Overview of WAA case study

NUWC chose the WAA in-board electronics application for two reasons. First, reengineering the WAA system contributes to the incremental insertion of commercial off-the-shelf equipment (COTS) into submarine warfare systems [2]. Second, a new implementation for the WAA inboard electronics suite would fulfill the Navy's need to reduce cost, as well as align with the Director of Defense Research and Development's (DDR&E) thrust areas. Two of DDRE's seven thrust areas emphasize *Affordability*, and *Sea Control and Undersea Superiority* [3]. Therefore, the reengineering of the existing WAA system responds to changing cost and commercial technology requirements.

This paper provides a brief description of the Wide Aperture Array full detection system. As interest arises, the approach will incorporate diversified case studies based on other existing Naval systems.

The cost effective implementation of the in-board electronics for a Wide Aperture Array full detection system serves as the primary objective for this case study. The WAA system can perform the detection function for a submarine. Sea test data indicates the Wide Aperture Array system can provide an attractive capability. In addition, current fiscal requirements within the Navy (i.e., the New Attack Submarine Program) have created the need to significantly reduce the cost of the existing WAA system.

The WAA full detection concept conceived by DePrimo and Choinski, and published in "An Efficient Approach to Systems Evolution (EASE)" [4], offers a cost effective way to implement the inboard electronics with additional full detection capability. This implementation serves as the case study for this paper.

The implementation uses commercial massively parallel processing technology developed within the High Performance Computing and Communications (HPCC) Program and made available under the Director of Defense Research and Development's (DDR&E) Defense Modernization Plan.

Figure 1 illustrates the WAA signal processing specified for the reengineering process. The signal processing includes additions to the existing WAA system.

This paper proposes a generic approach to mitigate the risk when investing in a massively parallel architecture. The approach concentrates on the importance of constraints and sanity checks throughout the design process.

3: System engineering approach

Figure 2 embodies the intent of Rechtin's heuristic, as quoted at the beginning of the paper, by setting up a process to assess the compatibility of the architecture and

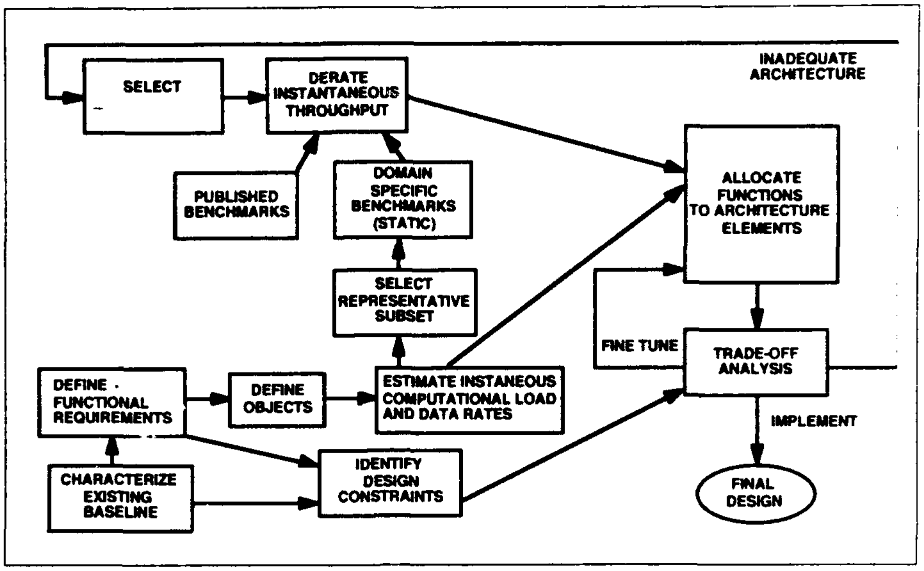

Figure 2. Systems Engineering Design Approach

the requirements throughout the design process. Each step progressively refines the assessment. The approach includes reengineering considerations when systems engineers pursue a new implementation for cost or technology upgrade reasons.

The process starts with the definition of functional requirements and the selection of a candidate architecture. For reengineering problems like the Wide Aperture Array, the process includes a step to characterize the existing system. The existing system characterization provides the baseline for the tradeoff analysis.

An object oriented software design follows the functional specification. The inclusion of the object oriented design step translates functional requirements to objects suitable for software design. This step will determine if object oriented design facilitates software portability and reuse. In practice, a systems designer could bypass this step in favor of functionally based software design.

The sustained throughput and data rate estimates follow the object oriented design. The throughput and data rate estimates enable a preliminary architectural sizing using the performance data from existing libraries or static benchmarks. Static benchmarks provide single processor performance data for metrics like efficiency. Therefore, the architecture sizing obtained at this point allows for an initial assessment of the instantaneous throughput levels quoted by manufacturers.

Given the preliminary architectural sizing, the systems designer can perform a detailed analysis of the architectural requirements for the given application. The detailed analysis consists of a combination of modeling, simulation and dynamic benchmarking.

Dynamic benchmarking entails the implementation of a processing subset on a scaled down MPP architecture. In this manner, dynamic benchmarking reduces risk. The move from using a single processor to multiple processors differentiates dynamic benchmarking from static benchmarking. Dynamic benchmarking also introduces partitioning, input/output (I/0) issues, and event driven processing attributes.

In addition, the dynamic benchmarks validate the detailed architecture models simulated in this step. The concept of using modeling, simulation and benchmarking for architecture validation was first introduced by Muñoz of the Naval Undersea Warfare Center [5]. Figure 3 elaborates on the allocation process identified in figure 2.

After allocating the functions, the final tradeoff analysis uses a set of previously defined metrics to compare the performance of the proposed implementation to the existing baseline system. The results of the tradeoff analysis determine whether to accept, modify or eliminate the candidate architecture.

3.1: Metrics

The basis of the tradeoff analysis rests with the extraction and comparison of metrics. Modeling and simulation permit measurement of the metrics for the proposed system. The measured data can be compared to the existing system baseline data.

Numerous metrics have been identified for consideration in the tradeoff analysis of MPP

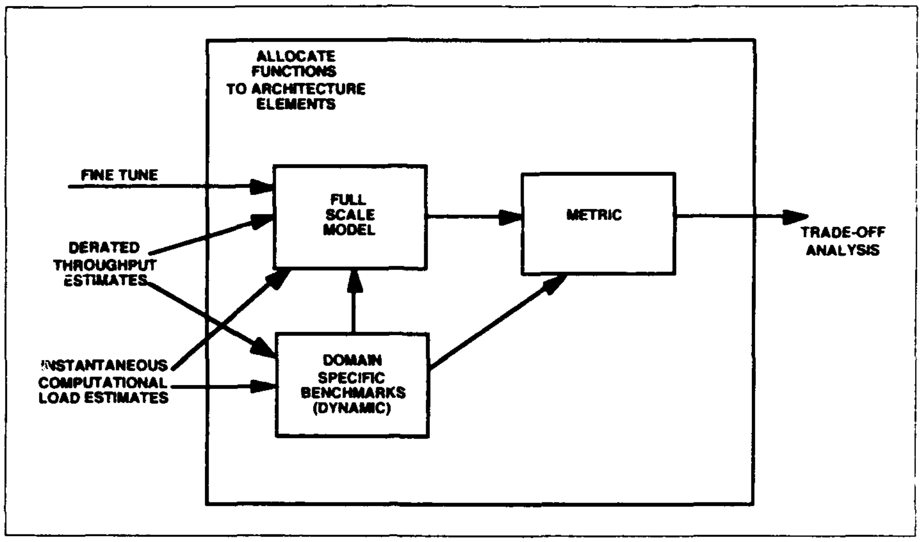

Figure 3. Functional Requirements Allocation

architectures. Table I presents the MPP metrics and their definitions. These metrics have been discussed in detail by Lee [6] and the team of Sweetman and Muñoz [7]. The design capture view metrics outlined by the ECS research block can also be added to this general list.

4: High level architectural sizing

The high level architectural sizing provides the preliminary estimate for the size and configuration of a compatible architecture. The high level architectural sizing consists of four parts:

1. capturing functional requirements,
2. baselining the existing system,
3. generating an object oriented design,
4. establishing a preliminary architectural sizing from static benchmarks.

Once completed, these four steps lead to a detailed architectural design and development. Unlike the detailed design, the high level architectural sizing does not address software issues or partitioning of the functions.

4.1: Functional requirements definition

The functional requirements definition phase of the generic system engineering design approach results in system level specifications for the application. For the case study highlighted in the paper, a systems engineer first documented the WAA functional requirements previously depicted in figure 1.

A systems engineering design tool like RDD-100 can capture the functional requirements and facilitate traceability throughout the design process. Initially, a word processor was used to capture the WAA requirements; however, NUWC will also use tools like RDD-100.

RDD-100 brings several capabilities to the design process including: requirements capture, functional behavior modeling, full scale architecture modeling, resource allocation, dynamic analysis and documentation of results. Other tools are also available to provide this capability.

4.2: Baseline system characterization

Ideally, the systems design engineer should baseline the existing system using metrics necessary to complete the tradeoff analysis. Under these conditions, the designer completes the tradeoff analysis by comparing the new and existing systems on equal footing.

Unfortunately, even the best documentation from a military system will fall short of supplying all the previously defined metrics for the tradeoff analysis. Design engineers document their work for development and not reengineering purposes. Therefore, the tradeoff analysis will embody comparisons between similar but not equivalent metrics.

The reengineering process was initiated by using the design capture views established by the ECS Block to capture the implementation of the existing WAA

Table 1. Metrics

1.	**Computation Bandwidth**	A description of the frequency of operations per unit time measured in MFLOPS/second.
2.	**Communication Bandwidth**	A description of the I/O rate measured in MBytes/second.
3.	**Memory Bandwidth**	A measure of the memory access requirements per unit time represented by Bytes/second.
4.	**FLOPS-I/O Ratio**	A ratio which compares the computation load(MFLOPS) to the I/O (Bytes/sec) load.
5.	**Latency- FLOPS Product**	A characterization of the ability to support communications requirements versus the computational bandwidth requirements of a module or architectural element.
6.	**Power/Weight/Volume**	Values used to characterize the physical attributes of a system. Power is characterized by Watts, weight by pounds (lb) and volume by cubic feet (ft^3).
7.	**dB/Watts**	A measure which combines process gain (dB), algorithm efficiency, dB/gate-Hz, technology cost, gate - Hz/watts, architecture efficiency, and percent duty cycle. An alternative is to use noise recognition differential (NRD) instead of process gain for a measure of sonar system performance.
8.	**Architecture Diameter**	An integer which represents the maximum number of communication paths that a message or data may be required to travel from processor to processor.
9.	**Architecture Latency**	The maximum time, in seconds, a message takes to propagate across the path that determines architecture diameter.
10.	**Processor Memory Ratio**	A ratio that captures the memory available to an individual processor. For local memory systems the ratio would be the local memory per processor. For shared memory systems the ratio would be computed by dividing the total system memory by the number of processors and adding the amount of local cache memory per processor.
11.	**Average Message Size per Processor**	A value computed by dividing the total number of message bytes sent during the time it takes to execute an algorithm, divided by the number of processors.
12.	**Response Time**	The time in seconds that is required to execute an algorithm. The time begins when the first processor starts executing and ends when the last processor stops executing.
13.	**Processor Utilization**	A percentage computed by dividing the sum of the individual times that the processors are executing by the total time it takes to execute the algorithm, times the number of processors in the system. $= \frac{t1 + t2 + t3 + \ldots tn}{NT}$
14.	**Program Size**	The size in bytes of the program.
15.	**Speed Up**	A value computed by dividing the response time for an algorithm executing on a single processor by the response time for an algorithm executing on several nodes in a system.

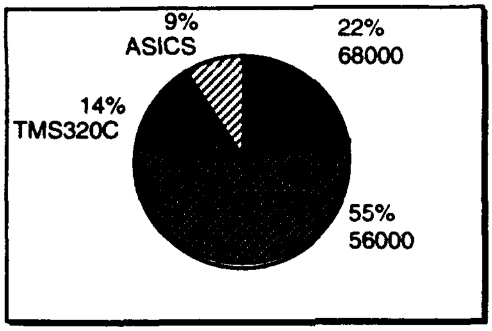

Figure 4. Processors

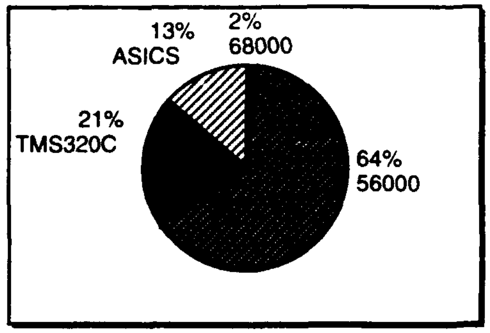

Figure 5. Throughput

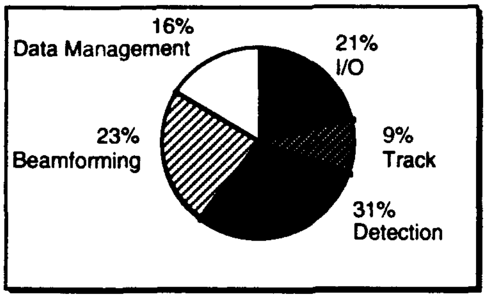

Figure 6. Processor/Function

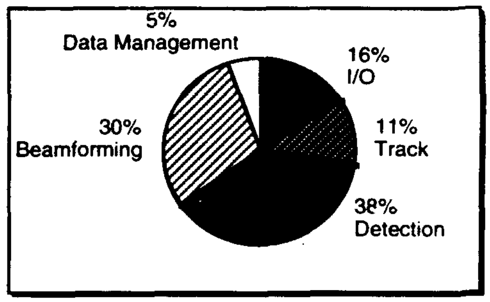

Figure 7. Throughput/Function

system. The information presented in figures 4-7 illustrates the types of data documented for the existing system. These figures represent a sample of the data used for the baseline characterization of the Wide Aperture Array System case study.

Although the existing baseline uses a distributed processing architecture, some of the experiences can be carried over to the massively parallel processor architecture. For example, since trackers do not require large amounts of throughput, the MPP implementation for trackers probably would not change significantly. Figures 4-7 present four different views of the Wide Aperture Array System. Partitioning functions to resources has become the focal point for the case study because of its significance in massively parallel array architectures.

4.3: Object oriented design

The object oriented software design follows the functional specification, and the existing system baseline. The inclusion of the object oriented step translates functional requirements to objects suitable for software design. Object oriented design should facilitate the reuse and portability of software.

Once the functional requirements have been designed, a software engineer determines the set of software objects necessary to achieve the desired functionality. An analysis takes place with the assistance of an object oriented design tool like OMTool. Future versions of products like OMTool will perform the functional to object oriented translation automatically; however, OMTool cannot perform the translation at this time.

OMTool provides functional, object and data flow views for a given application. In addition, the tool produces C++ code. This paper neither endorses nor denounces the use of OMTool. Engineers working on the WAA case study use OMTool because of the features available for the given price range.

Figure 8 illustrates the object oriented array system design. Figure 9 expands the object oriented beamformer design. These diagrams represent a synopsis of the object oriented design which will be used to reengineer the WAA system. Note that although the object oriented design started with the WAA application in mind, the high level software suits any array processing problem using a 2 stage time delay beamformer.

In the future a systems engineer specifying the functional level requirements could expedite the object oriented design if a link was developed between tools like OMTool and RDD-100. The link could further automate the design process. In addition, the link would also ensure the consistency of requirements between object oriented and system design tools.

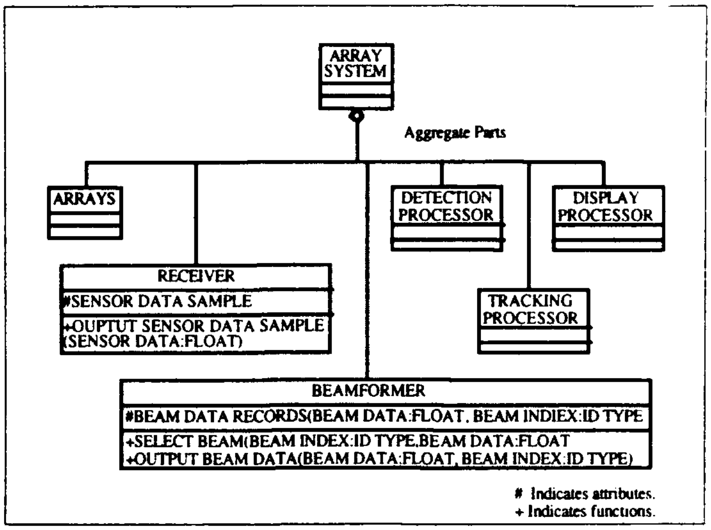

Figure 8. Object Oriented Array System Design

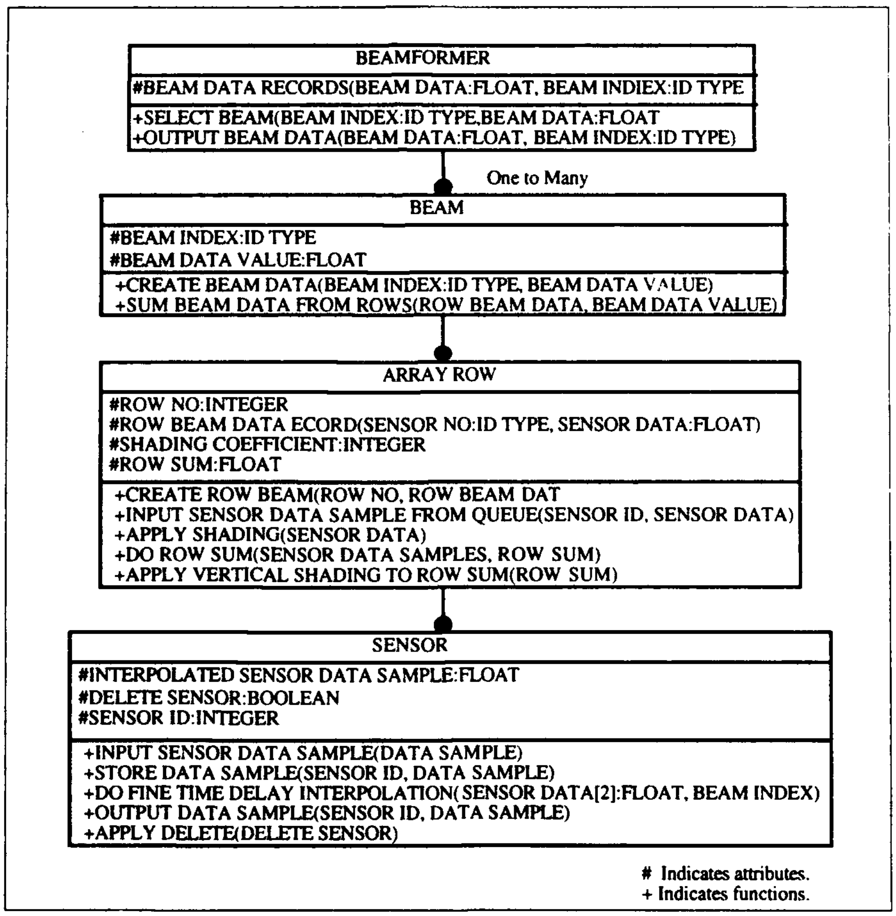

Figure 9. Object Oriented Beamformer Design

4.4: Instantaneous Load Estimation and Static Benchmarking

A preliminary sizing for the WAA case study demonstrates the application of instantaneous load estimation and static benchmarking. The number of floating point multiply and addition operations were calculated for the functions identified in figure 1. The sustained throughput estimates in Table II reflect these multiply and addition estimates coupled with input data rates.

Intel Corporation provided the efficiency and peak numbers in Table II based on Paragon single processor implementations written in Fortran. The peak numbers do not reflect scaling effects due to I/O and partitioning. As a result, efficiencies for a final massively parallel version would probably be lower. Therefore, Table II presents the results of a static benchmarking effort and represents a preliminary sizing for the WAA processing problem. Basically, initial estimates indicate the WAA processing requires a massively parallel architecture capable of providing 36 GFLOPS of peak throughput.

Table II. Static Benchmarking Load Estimation [8]

Operation	Sustained	Efficiency	Peak
Beamforming	7.50	32%	23.44
FIR Filters	0.54	25%	2.16
Complex FFTs	2.02	56%	3.61
Cross PSD	0.23	14%	1.64
Auto PSD's	0.16	14%	1.14
Integrate Auto Spectra	0.01	14%	0.07
Inverse Complex FFT	1.01	56%	1.80
Normalized XCOR/ Up Sample and Interpolate FIRs	0.25	14%	1.79
Total GFLOPS	11.72		35.65

5: Detailed modeling and requirements allocation

The detailed level modeling and requirements allocation method provides a specific design for a MPP architecture. The method presented in this paper addresses four issues:

1. technology independence,
2. software partitioning,
3. full scale modeling, and
4. dynamic benchmarking.

Technology independence means that it is possible to retarget the software. Partitioning involves dividing the processing into pieces which can run on individual processing elements.

Massively parallel architectures can have an assorted collection of heterogeneous analog or digital processors. The program that runs the real-time embedded system typically can have hundreds of thousands of lines of source code. The system is generally very complex, difficult to design, and hard to maintain.

Large combat systems historically use a number of heterogeneous processors connected in a distributed network structure. Continuing this trend would lead to expansive custom MPP architectures.

Custom MPP architectures have thousands of processors connected as nodes in some kind of network structure. Commercial processors like the Intel i860, Sun Sparc, or DEC Alpha chip perform the processing functions in the nodes. Hypercube, mesh, hierarchical ring, or tree topologies form the basis of the networks.

Companies like Intel, Thinking Machines and Kendall Square Research have developed the MPP architectures into commercially available systems. These systems may have homogeneous or heterogeneous processing elements. The difference between commercial and custom MPP architectures lies in the user base. System engineers optimize custom architectures for one specific application for a limited market. Companies build commercial architectures as products for a more generic user community. The commercial architectures may not fit a particular application as well as a custom architecture; however, the commercial architecture will fit a broader range of applications. Table III characterizes custom and commercial architectures.

Table III. Custom Versus Commercial MPP Architectures

MPP Architecture	H/W Cost	S/W Cost	Portability
Custom	high	high	none
Commercial	medium	medium-high	partial

If MPP markets develop successfully, the hardware cost of commercial MPP architectures will shrink faster than custom MPP architectures. The software development costs of the custom architectures are

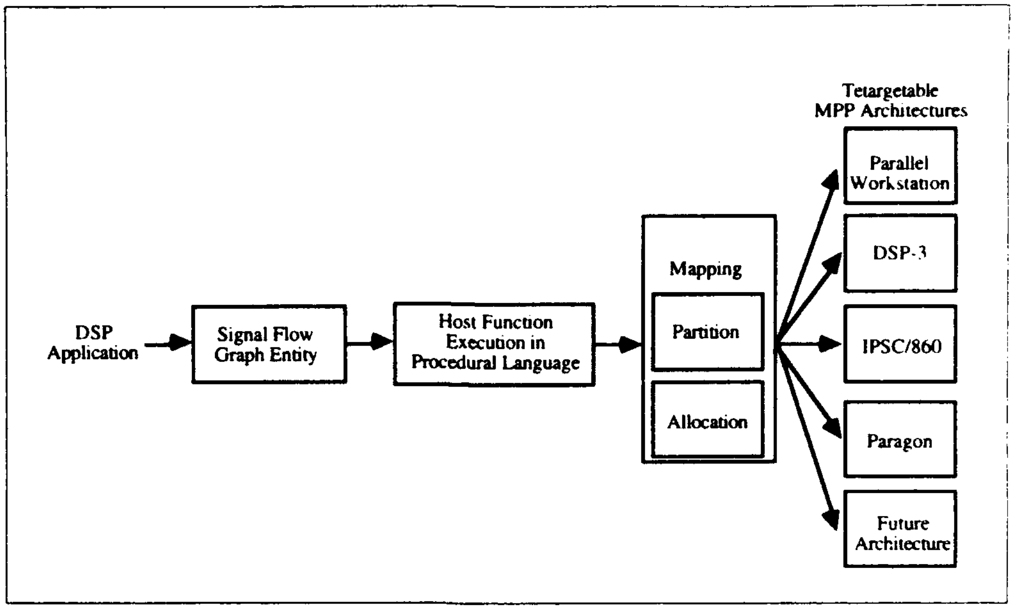

Figure 10. Technology Independent Application

prohibitively high. Life cycle costs for custom architectures are also high because of lack of portability. With appropriate research and development in the engineering of complex systems, the software for commercial MPP architectures can achieve lower cost through partial portability. One objective of the Massively Parallel System Design task is to address detailed level MPP software mapping and portability.

Despite continuing research efforts in parallel processing, two challenges exist for MPP architectures:

1. The MPP scalability problem presents a major obstacle. Efficiencies from benchmarks with large (thousands of processors) MPP architectures measure less than 10%. For vector processors like the Cray supercomputer, the efficiencies measure higher than 10%. These inefficiencies create a high incentive to increase the speed-up of MPP systems.

2. The MPP programming problem necessitates a significant up front development effort for partitioning. Software engineers cannot program MPP architectures easily. One dominating issue relates to the mapping process. The mapping process determines partitions and allocates functions on MPP architectures. The absence of automated mapping tools requires software engineers to manually complete the mapping process.

Scalability and partitioning are correlated. Good software partitioning methods generally lead to good scaling. Generally, the efficiency and scalability increase with effective software development techniques. Note however, that this relationship is not linear and is algorithm dependent.

5.1: Technology independence

Technology independence presents a significant hurdle to real-time embedded MPP architectures. Figure 10 shows one approach for attaining technology independence. In general, the objective and procedures are similar to other previous works. The uniqueness lies in the details of the methodology. The method concentrates on using commercially available tools whenever possible. Many of these tools have graphical user interfaces.

Graphical interfaces facilitate the use of signal flow graphs for representing real-time embedded applications. Node labels represent computation loads in the signal flow graph. Directed edges symbolize data dependency in the graph. Edge labels characterizes the communication delay of signals from node to node. Simple FIFOs between the nodes can represent communication delays for some target architectures.

This kind of programming method uses block diagrams, large grained data flow graphs, and synchronous data flow graphs. The graphics facilitates the entry of a digital signal processing (DSP) application. The task

software may be written in any procedural language so that simulation of the function can be done on host processor before it is mapped into a target MPP system.

This method meets the scalability and portable software challenges. Software engineers can use one of three different techniques to program MPP architectures. The first one takes a regular sequential program and compiles it for a MPP system. This technique is referred to as the parallelizing compiler approach. The second recodes the program in a parallel language such as LINDA, FORTRAN 90, or functional (applicative) language. This technique is called parallel languages. The first technique does not require a large effort when rewriting software. A parallelizing compiler capable of dealing with thousands of lines of code simply does not exist, and the ones available for small programs suffer from performance problems. The second approach requires a new culture for programmers. However, using parallel language still falls short of acceptable performance.

The third approach follows the message passing methodology which involves explicit parallel environment control. Hence, the third technique is called message passing. The programming takes place in an environment like PVM or EXPRESS with utilities to handle parallel message passing. The last technique requires some user awareness of the topology of the MPP architecture, but it can achieve the highest scalability and efficiency. Table IV describes the software techniques.

Table IV. MPP Software Approaches

Technique	Efficiency	Mapping
Parallelizing Compiler	Not Proven	Automatic
Parallel Language	$\leq 0.01\%$	Automatic
Message Passing	1% ~ 10%	Manual

Unfortunately, automatic mapping technology for partitioning and allocation does not exist. Good performance in programming MPP architectures relies on tedious manual mapping methods.

Single Instruction Multiple Data (SIMD) MPP and the Multiple Instruction Multiple Data (MIMD) MPP architectures further complicate the portability challenge. SIMD MPPs encompass the connection machine and the MASPAR architectures. MIMD MPPs consist of the iPSC 860, CM-5, DSP-3, and Paragon shown in Figure 10. This paper concentrates on MIMD architectures.

The four salient features for the portable massively parallel systems design (MPSD) method discussed in this paper include:

1. task level function module parallelization (coarse grain),
2. high level procedural language and messaging passing,
3. portable software for different MIMD MPPs, and
4. calibrated performance metrics for mapping.

After graphic entry and host function simulation a mapping procedure is required before the program can run on a MPP system. Figure 10 shows a set of MPP architectures. MIMD with distributed memory and message passing are our target systems. The Mapping Procedure consists of partition and allocation.

For MIMD with distributed memory and message passing, scheduling may be done at the compile time. Unlike real-time scheduling, compile time scheduling is the most straightforward way to handle the real-time requirement which dominates military applications.

5.2: Partitioning

The Calibrated mapping performance prediction paradigm (CMPP) leads to detailed modeling allocation. The CMPP paradigm is discussed in this section of the paper.

Because of a large and multidimensional solution space, heuristic methods provide the first pass solution. Therefore, automation and design aids would expedite a broader search of a good solution for the total system design.

Technology independence depends also on the MPP mapping procedure. Mapping involves partitioning and allocating function modules on the MPP architecture. The absence of parallelizing compilers and languages leaves only design aids to ease the mapping process. The user has to couple the procedural modules with message passing operations. The process is slow when the user has to do a manual mapping for all the pieces (thousands), as well as run the MPP execution to decide whether the mapping works. Figure 11 shows this mapping procedure in detail. This cycle will be repeated to optimize each performance metric.

This paper proposes a calibrated mapping performance prediction paradigm. Figure 12 illustrates the paradigm. The key idea concentrates on performance model simulation. Rather than do a functional execution on full scale MPP to collect dynamic performance metric data, the benchmark is collected from model simulation. The full scale model then provides estimates of the architectural performance in terms of the previously defined metrics.

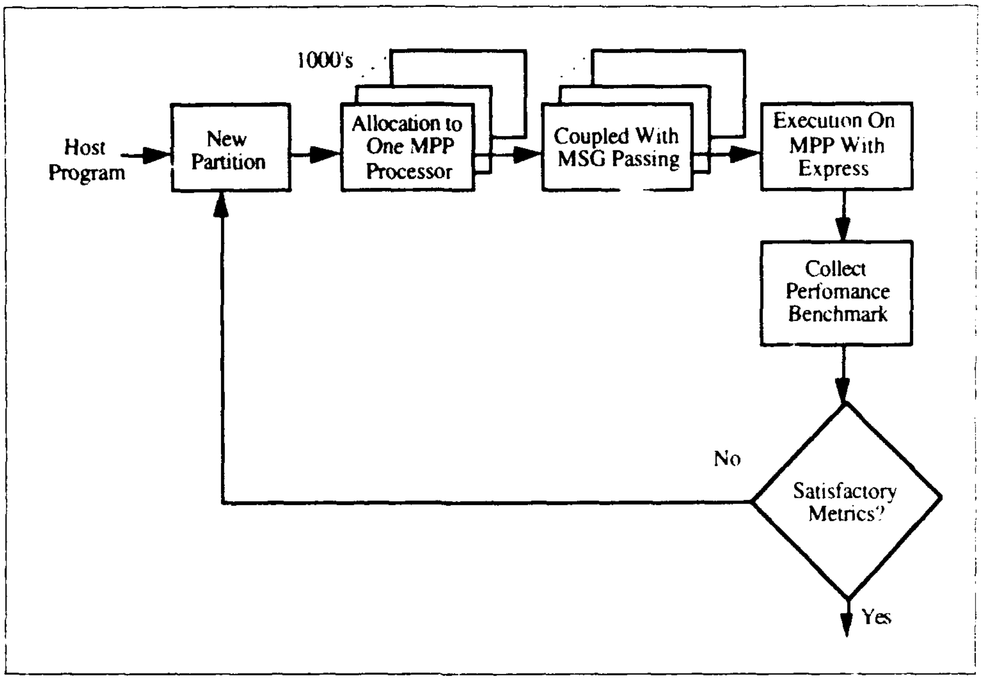

Figure 11. Detailed Mapping Procedure

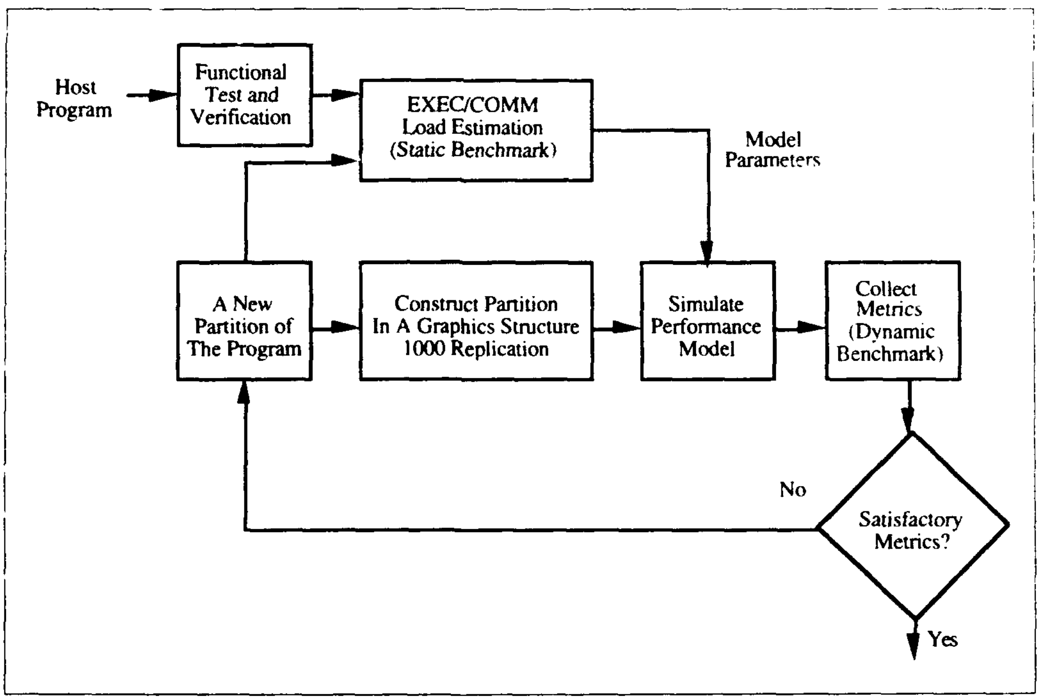

Figure 12. Calibrated Mapping Performance Prediction Paradigm

5.3: Full scale modeling

The performance model requires two kinds of modules: the execution module (EXEC), and the communication module (COMM). The execution time metric as the initial focus, since execution time is directly related to the speed up in MPP architectures. EXEC load, EXEC bandwidth, COMM load, and COMM bandwidth characterize these modules. The host program estimates the EXEC load and COMM loads for all the partitioned pieces of a specific mapping. The collected data become model parameters to annotate the performance model before simulation. Each new partition requires repetition of the load estimation and extraction process. Any automation that can be added would be desirable.

EXEC modules and COMM modules are used to build the performance model with token networks. The token network handles multiple transmitters like real network situations. Presently the model can only handle Ethernet simulation. Construction of the performance model is done in the graphics mode. The VHDL feature simplifies the replication of thousands of identical modules. The Calibrated Mapping Performance Prediction (CMPP) paradigm hides many of the details of message passing so that the designer can concentrate on the partition and allocation problems. The right environment enables replication the modules many times. This environment reduces the problem of scaling to thousands of processors.

The CMPP paradigm discussed in this paper used the VHDL environment. Note that VHDL is not used here for hardware design; instead VHDL allows the designer to construct the structure, simulate the performance, and collect metric data. Both PC's and workstations support VHDL environments at low cost. VHDL will be available for hardware and system design for a long time. In addition, constructs of the VHDL language can replicate modules as shown in Figure 12 in a straight forward manner. VHDL *generic* constructs also help annotate model parameters before simulation. The manual EXEC/COMM load estimation and extraction is a disadvantage of the CMPP paradigm. An automatic procedure would strengthen the CMPP paradigm.

The CMPP paradigm allows the partition and allocation results to be portable to different types of MPPs. Remapping is necessary due to different network bandwidths, topologies and throughput rates in different MPPs. However the CMPP minimizes the portability effort as much as possible. The CMPP paradigm reduces the effort needed to run a real-time embedded application on different MPP architectures.

The FLOPS-I/O ratio characterizes the proportion of computation done versus communication (I/O) required in the partition. The ratio can characterize the architecture element once maximum throughput requirements are fulfilled. If the peak FLOPS-I/O ratio of an architecture element is less than that of the application module, it is possible to fit the application module into the element. If the peak FLOPS-I/O ratio of an element is greater than the application module, the partition will encounter problems.

Essentially, the FLOPS-I/O ratio characterizes computational activities relative to communication activities. With this metric, it will be easier to analyze the results of different mapping processes by examining granularity. One definition of fine grain tasks refers to small FLOPS-I/O ratios. Fine grain application modules can only be efficiently accommodated in fine grain architecture elements.

The FLOPS-I/O ratio metric makes it possible to find a common partition of an application for a set of MPPs. The common partition usually can not achieve the best speedup and efficiency in a specific MPP, but the partition can be accommodated in a number of MPPs. Further development of the CMPP paradigm will demonstrate this situation in the future.

A collection of Sparc workstations on an Ethernet was used to demonstrate the CMPP approach during 1993, since the researchers did not have access to a commercial MPP architecture. The researchers also used a message passing development environment called EXPRESS. EXPRESS addresses the portability challenge for the CMPP paradigm.

The development consists of three parts. First, the EXEC module characterizes a piece of the execution that occurred in the architecture element of the MPP. EXEC modules represent a source that generates a load token, a feed-through that accepts input tokens and produces output tokens, or a sink that consumes a load token. The following VHDL parameters characterize EXEC modules:

INST => unique module name
Unit => 1 Kbytes
Size_info => statistic size in units
Throughput_info => (#/sec) statistic throughput rate
Latency_info => statistic delay (usec.)

- * Duty_cycle_info=>(#/sec) statistic duty cycles
- * Only relevant for source EXEC modules.

Figure 13 shows a VHDL structure for the modules. The left most block depicts a source EXEC module, and the right most one a sink EXEC module. The INST generic describes a unique name for the module in the model. The size *unit characterizes the load of execution.

The term "unit" represents the basic data size such as in bytes or Kbytes. The throughput characterizes the speed of this EXEC module. The latency feature permits a more accurate delay account. Duty cycle is relevant if the EXEC module is a source that generates periodic loads.

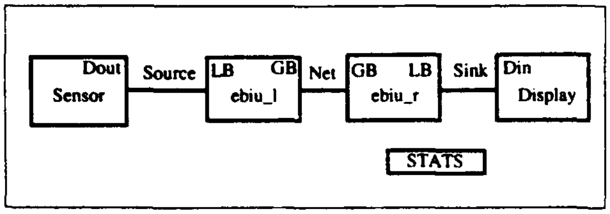

Figure 13. A Structure of EXEC Modules, COMM Modules, and Ethernet

Figure 13 shows two COMM modules called ebiu. The COMM module can receive or transmit to or from a local port. The data transfer on the glob-- -:t is also bi-directional. The following VHDL generics characterize the COMM module:

INST => unique module name
bw_unit_per_sec => unit size (byte)
bandwidth_info => statistic bandwidth (byte/sec)
Tx_latency_info => statistic transmit latency delay (usec)
Rx_latency_info => statistic receive latency delay (usec)
Bus_timeout_info => statistic time-out (usec)
Ack_time_info => statistic acknowledgment time (usec)

Bandwidth and bw_unit_per_sec characterize the channel limitation of the bus. For the case of Ethernet, part of the Ethernet features reside in the COMM module, and the other features like arbitration reside in the token signal resolution function.

The VHDL resolution function is a special facility available in the VHDL language that handles multiple signal drivers. The signals in this model are all data token types. A special VHDL resolution function is implemented to model the Ethernet.

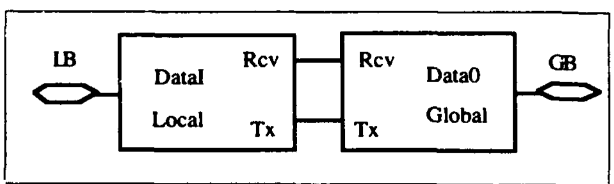

Figure 14. Ebiu Sub Module Structure

The ebiu is in turn built from two sub modules: Local and Globalnet. The sub module structure is shown in Figure 14. The VHDL environment can build these entities, module structures, and sub module structures hierarchically. Graphics windows permit editing, checking, and simulation. The bottom level behavior of the EXEC or COMM modules are written in VHDL.

5.4: Dynamic benchmarking

Dynamic benchmarking helps to validate the full scale models. The data from the dynamic benchmarks help refine the simulation models to reduce the risk associated with the scaling process. Due to the lack of availability of a commercial MPP architecture during 1993, the Naval Postgraduate School researchers explored the CMPP paradigm using Sun Workstations connected by Ethernet.

One important feature in the CMPP paradigm involves the calibration process for EXEC/COMM model parameters. The calibration process requires dynamic benchmarking for fine tuning. The EXEC/COMM parameters are extracted from a functional program. The results enable calibration of the model. The calibration process refers to the adjustment of parameters by comparing a benchmark from the CMPP prediction to the parameters from the actual execution. The calibration process ensures the validity of the model.

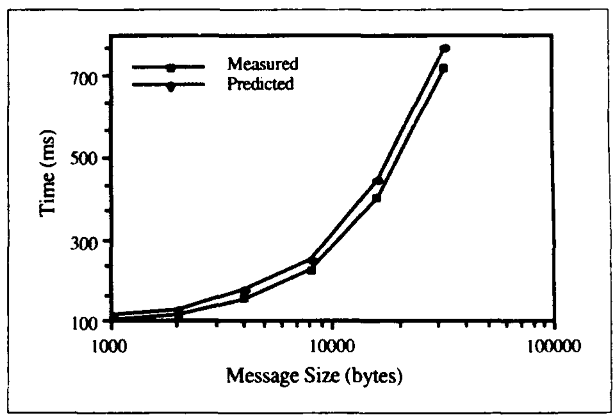

Figure 15. Ethernet Delay for Versus Message Size

The crucial step for the experiments developed during 1993 was to model and characterize the Ethernet correctly. The aforementioned calibration process tuned the COMM modules (ebiu). Figure 15 presents the actual message delays and the model predictions. The message size varied from 1 Kbyte to 32 Kbytes. The predicted and measured data matched very well. The model parameters that yielded this prediction consist of:

bw_unit_per_sec => byte
bandwith_info => 48,000
Tx_latency_info => 41.280 ms
Rx_latency_info => 10.000 ms
Ack_time_info => 41.280 ms
bus_timeout_info => 10 sec

In addition to the Ethernet modeling, two beamformers were coded and tested. One beamformer used a frequency domain algorithm, and the other a time domain algorithm. The time domain algorithm reflects the type used in the Wide Aperture array system.

The frequency domain beamformer demonstrated the advantage of using MPP systems. The hypothetical beamformer assumed 96 sensors in the system. Beam response covered 0 to 180 degrees with 1 degree resolution. A host program in FORTRAN 77 was written and checked with the test data to assure correctness. The mapping procedures outlined in this paper were used to partition and execute the application under the parallel EXPRESS environment. The metric plotted in figure 16 is the execution time. This mapping procedure was repeated for 1, 2, 4, 6, and 8 architecture elements on the network of workstations. The results show that increasing processing elements decreases the execution time.

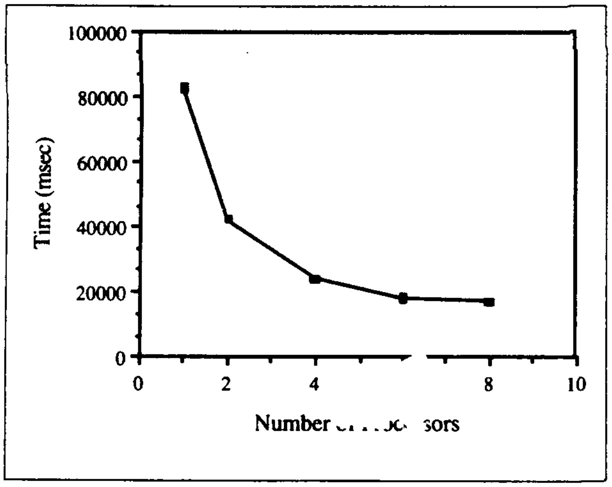

Figure 16. Execution Time Versus Number of Processors

Figure 17 exhibits the computational and communication loads for the frequency domain beamformer. The loads were estimated and extracted as described in the CMPP paradigm in Figure 12. These estimates represent the loads for each processor.

The two main execution modules are: the FFT module and the Vector-Matrix product module. The other modules are executed in the host. The diagram shows that the largest execution load occurred in the Vector-Matrix module. The heaviest traffic on the Ethernet was the message shuffle between the FFT and Vector-Matrix modules. The information in figure 17 was accumulated using the Sun operating tcov command. COMM loads were estimated using EXPRESS profilers.

The parallel EXPRESS environment can also provide an event profile which shows the communication activities, and the execution activities of the processors.

After the analysis, the next step is to construct a partition structure in the VHDL environment that simulates the performance. A structure for the 8-node partition was developed. The objective is to be able to predict performance such as execution time shown in figure 16. Progress is ongoing and encouraging.

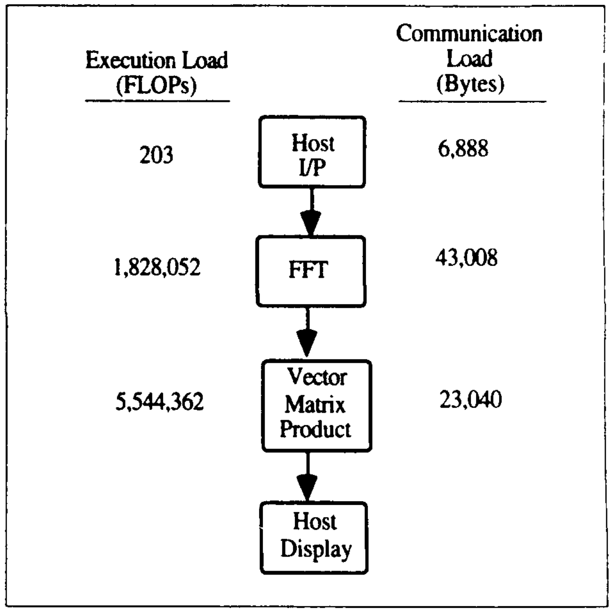

Figure 17. EXEC and COMM Loads for an 8-Processor Partition

A single panel of the WAA beamformer was also programmed during 1993. The WAA program includes test data generation, time delay memory, 1:3 interpolation, full beam vertical shading, and beam summation. The program has been tested and verified.

EXPRESS will be used to map the application to the Sun Workstation environment. Table V reveals preliminary execution time data for the WAA beamformer program on three high speed computers: the Sparc 630MP (2 processors), the Navy TAC-3 (HP 900/730), and the Cray YMP/EL. The Cray yielded the best execution time, but the TAC-3 yielded the smallest execution code size.

The TAC-3 is about 10 times slower than the Cray YMP/EL, but requires 25 times less code.

TABLE V. Time Domain Beamformer Benchmark

Architecture	Execution Time (sec)	Execution Code (Bytes)
Sparc 630MP (2 processors)	833.8	237,568
TAC-3 (HP9000/730)	339.6	32,768
Cray YMP/EL (4 processors)	35.4	802,320

6: Summary

This paper proposed a generic approach to mitigate the risk when investing in a specific MPP architecture. The approach proposes steps to assess the architecture and requirements compatibility which include: defining the system requirements, sizing an architecture using static benchmarks, allocating resources using systems engineering tools, developing a full scale model, validating the full scale model with dynamic benchmarks, assessing the compatibility of the architecture with the real-time embedded application, and selecting the appropriate design approach based on a trade-off analysis. The approach embodies reengineering considerations for cost or technology upgrades.

The Wide Aperture Array case study demonstrated the approach's usefulness. This paper presented a functional specification, existing system baseline, object oriented design and a series of benchmarks for the WAA application.

In addition to the WAA case study, the paper also included detailed modeling and simulation data for a frequency domain beamformer implemented on several Sun Workstations networked with Ethernet. This test demonstrated the utility of the Calibrated Mapping Performance Prediction Paradigm. The information presented included an explanation of the detailed VHDL models that a researcher fine tuned to reflect the actual operation of the network.

Several issues surfaced during the course of the research described in this paper. These issues offer possibilities for extended research in the future. The issues include:

1. compatibility between engineering tools (e.g., RD-100, OMTool, etc.),
2. portability of object oriented design software,
3. technology independent software for massively parallel architectures,
4. scalability of massively parallel architectures,
5. availability of commercial massively parallel architectures, and
6. suitability of the massively parallel systems design approach to other case studies.

Research will continue to pursue the massively parallel system design framework discussed. Plans encompass implementation, modeling, benchmarking and simulation of the Wide Aperture Array functions. A continued focus will be placed on using tools like VHDL, RD-100 and OMTool, in addition to integrating tools emerging from the Engineering of Complex Systems Block Program.

Acknowledgment

The authors acknowledge several individuals for their significant contributions to this paper: Daniel Organ for the characterization of the existing WAA system, Warren Axtell for the object oriented design, and Adam Siconolfi for the WAA functional requirements specification. In addition, the authors would like to thank Dr. José Muñoz for his helpful comments and suggestions, and John DePrimo for providing the technical review of the paper.

References

[1] Eberhardt Rechtin, Systems Architecting: Creating and Building Complex Systems, Prentice-Hall, 1991, p.48.

[2] Thomas C. Choinski, "Economical Development of Complex Computer Systems," Proceedings of the Complex Systems Engineering Synthesis and Assessment Technology Workshop, July 1992.

[3] Director of Defense Research and Engineering, Defense Science and Technology Strategy, July 1992.

[4] Thomas C. Choinski and John J. DePrimo, "An Efficient Approach to Systems Evolution," Proceedings of the Complex Systems Engineering Synthesis and Assessment Technology Workshop, July 1993.

[5] José L. Muñoz, et al., "An Architecture Assessment Environment for Massively Parallel Computations," Proceedings from the IEEE International Conference on Systems, Man, and Cybernetics, Volume II of III, November 14-17, 1989.

[6] Chin-Hwa Lee, "Massively Interconnected Models for a Beam Former," Proceedings of the 1992 Complex Systems Engineering Synthesis and Assessment Technology Workshop, July 1992.

[7] Denman E. Sweetman and José Muñoz, "Measures of Effectiveness (MOEs) for Parallel Architectures," Proceedings from the IEEE International Conference on Systems, Man, and Cybernetics, Volume II of III, November 14-17, 1989.

[8] Thomas C. Choinski and John J. DePrimo, "An Efficient Approach to Systems Evolution," Proceedings of the Complex Systems Engineering Synthesis and Assessment Technology Workshop, July 1993.